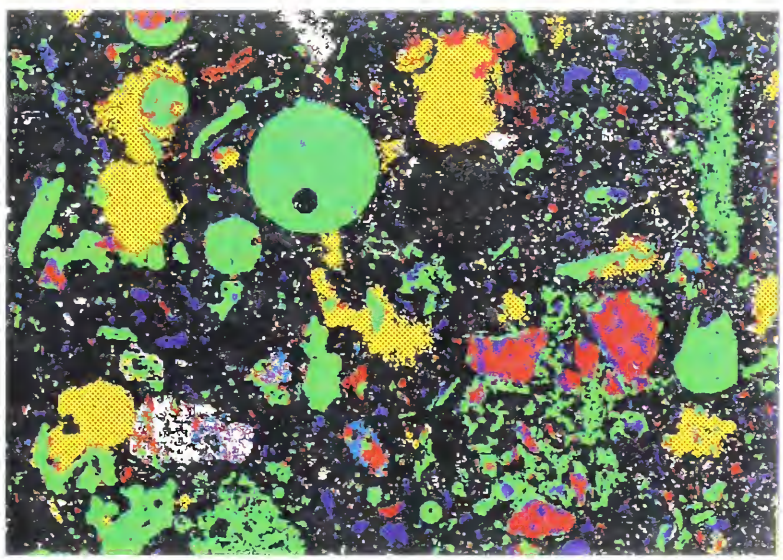
NISTIR 6050

Incorporation of Fly Ash into a 3-D Cement Hydration Microstructure Model

Dale P. Bentz
Sebastien Remond



Building and Fire Research Laboratory
Gaithersburg, Maryland 20899

United States Department of Commerce
Technology Administration
National Institute of Standards and Technology

NISTIR 6050

Incorporation of Fly Ash into a 3-D Cement Hydration Microstructure Model

Dale P. Bentz
Sebastien Remond

August 1997
Building and Fire Research Laboratory
National Institute of Standards and Technology
Gaithersburg, Maryland 20899

U.S. Department of Commerce
William M. Daley, *Secretary*
Technology Administration
Gary R. Bachula, *Under Secretary for Technology*
National Institute of Standards and Technology
Robert E. Hebner, *Acting Director*

ABSTRACT

Extensions have been made to the NIST 3-D cement hydration and microstructure development model to incorporate pozzolanic materials such as silica fume and fly ash. These additions will enable the simulation of the hydration and microstructure development of blended cements as well as conventional Portland cements. SEM and image analysis techniques originally developed for differentiating the cement clinker phases present in a 2-D image of cement particles have been modified and applied to equivalent images of fly ash particles. Reactions have been proposed and the appropriate volume stoichiometries determined for the various components of fly ash. Some preliminary calibrations of reaction rate constants and dissolution rates have been performed by comparison to experimental results for the adiabatic heat signatures of a variety of concretes with various water-to-cement ratios and fly ash contents. Further research needed to complete this calibration is outlined.

Keywords: Building technology, cement hydration, computer modelling, fly ash, microstructure, pozzolan, silica fume, simulation.

Contents

List of Figures

List of Tables

1 Introduction

This document describes the efforts to date for incorporating fly ash and silica fume into the NIST 3-D Portland cement hydration and microstructure development model [1]. A series of fly ashes (from the U.S. and France) have been imaged using a combination of SEM backscattered electron and X-ray signals to distinguish the major phases present (silica, an aluminosilicate, calcium chloride, tricalcium aluminate, a calcium aluminosilicate, anhydrite, and inert material). Computational programs for distributing the cement and fly ash phases amongst a size distribution of three-dimensional particles have been developed. The fly ash components may be distributed randomly amongst a subset of the particles, or as individual monophase particles. Reactions have been established for these starting materials and the phases of ordinary Portland cement based on information available in the literature [2, 3]. Volume stoichiometries and heats of reaction have been postulated for these reactions, which then have been included in a new version of the NIST 3-D microstructural model. Some validation against adiabatic heat release curves has been performed for one of the French fly ashes. The basic procedures are thus in place for modelling the hydration behavior of a variety of blended cements (and concretes), but further validation and adjustment of some constants (reactions rates, etc.) is still needed. A suggested approach for performing this validation is also presented.

The new version of the hydration model and the auxiliary programs provided in Appendix B are available via anonymous ftp from edsel.cbt.nist.gov (129.6.104.138). The user can log in as "anonymous" and supply their e-mail address as the password. The new programs can be found in the **/ftp/pub/CEMHYD3D/flyash** subdirectory. The programs have been developed and executed on a UNIX-based computer system and should require no more than 16 Mbytes of memory for execution. In addition, a postscript version of this documentation can be found in the **/ftp/pub/CEMHYD3D/flyash/manual** subdirectory. Complete documentation for the 3-D cement hydration and microstructure development program has been published previously [1] (also available in the **/ftp/pub/CEMHYD3D/manual** subdirectory) and should be consulted along with this update.

2 Experimental Procedures

SEM and X-ray analysis provide a convenient and powerful means for examining the microstructure of fly ash particles [2, 4]. Here, the SEM/imaging techniques developed previously [5] for imaging Portland cements have been modified for the analysis of fly ash samples. In addition to the backscattered electron (BSE) image, X-ray images are collected for Al, Ca, Cl, S, and Si. Based on the intensities of these signals, a separation into the following phases is performed: silica (SiO_2), an aluminosilicate ($SiAl_2O_5$), calcium chloride ($CaCl_2$), anhydrite ($CaSO_4$), a calcium aluminosilicate ($CaSi_2Al_2O_8$), tricalcium aluminate ($Ca_3Al_2O_6$), and inert (other) material [2, 3]. The initial segmented image is typically processed with a median filter [1] to improve the image quality by removing much of the noise within individual phase regions. A series of final processed images for a variety of fly ashes is provided in Appendix A of this document. These final images can be analyzed to determine phase volume fractions, phase surface area fractions, and correlation functions to be used in reconstructing representative three-dimensional images [1].

In reality, of course, the phase composition of the fly ash is more complex than being simply composed of the compounds listed above. However, in approximating the fly ash to be composed of these compounds, we are operating in much the same spirit as that which utilizes the Bogue compounds (C_3S[1], C_2S, C_3A, and C_4AF) as a representation of a Portland cement.

3 Three-Dimensional Reconstruction

Once the phase compositions, phase surface areas, and correlation functions have been determined, the same computational techniques developed for three-dimensional Portland cement particles can be used to reconstruct three-dimensional fly ash particles following a specific particle size distribution [1, 6]. In fact, using digitized spheres to represent the fly ash particles should in general be more realistic than in the case of cement particles, since fly ash particles are generally much more spherical in shape than cement particles. In addition to the computer programs for performing the reconstruction based on autocorrelation analysis, two new computer programs for distributing the fly ash phases amongst the fly ash particles have been developed. The first simply distributes the phases, in their appropriate volume fractions, totally at random amongst all of the pixels initially identified as fly ash. The second achieves this random distribution amongst the fly ash particles, so that all particles are monophase, but the user-supplied appropriate phase volume fractions are nonetheless maintained. This program has been developed because it has been observed in SEM images that the fly ash particles are often monophase. Listings for these two programs (**distfarand.c** and **distfapart.c**) are provided in Appendix B of this document.

During the assignment of phases to the fly ash particles, the tricalcium aluminate contained in the fly ash is assigned a different phase ID than that used for the tricalcium aluminate present in the Portland cement. However, within the microstructural model, the two are considered to be equivalent and participate in all of the same reactions, with the same dissolution probabilities and diffusion rates. While fly ash often replaces cement on a mass basis, for use in the model, this substitution must be converted to a volume basis. A specific gravity of 2.2 is generally used for the fly ash particles for performing this conversion. When using the program **genpart3d.c** (listing provided in Appendix B), the fly ash and cement can follow different particle size distributions, but the user should be sure to place all of the particles in order of largest to smallest, regardless of their phase identification.

4 Additional Reactions

The needed physical properties of the phases found in fly ash and the relevant hydration products are summarized in Table 1. While specific gravities, molecular weights, and molar volumes are readily available in the literature [3, 7], heat of formation data have yet to be located for all of the phases. Figure 1 summarizes the proposed reactions between the fly ash and the cement phases and their hydration products, such as CH. The numbers below each reaction indicate the volume stoichiometries (on a pixel basis) which must be maintained by the computer model. Based on these volume stoichiometries, all but one of the reactions

[1] Conventional cement chemistry notation is used throughout this document with $C = CaO$, $S = SiO_2$, $A = Al_2O_3$, $F = Fe_2O_3$, $H = H_2O$, and $\bar{S} = SO_3$.

in Figure 1 are seen to further contribute to the chemical shrinkage [6] occuring during hydration, as the volume of the solid hydration products is less than that of the reactants (solids and water). The only exception to this is the conversion of primary $C–S–H$ to "pozzolanic" $C–S–H$.

Table 1: Physical Properties of Proposed Components of Fly Ash

Compound Name	Compound Formula	Density (Mg/m^3)	Molar volume (cm^3/mol)	Heat of formation (kJ/mol)
Silica	S	2.2	27.	-907.5
Aluminosilicate	AS	3.247	49.9	
Stratlingite	C_2ASH_8	1.94	215.63	
Anhydrite	$C\bar{S}$	2.61	52.16	-1424.6
Gypsum	$C\bar{S}H_2$	2.32	74.2	-2022.6
Calcium chloride	$CaCl_2$	2.15	51.62	-795.8
Friedel's salt	$C_3A(CaCl_2)H_{10}$	1.892	296.66	
Calcium Hydroxide	CH	2.24	33.1	-986.1
Calcium silicate hydrate	$C_{1.7}SH_4$	2.12	108.	-3283
Pozzolanic $C–S–H$	$C_{1.1}SH_{3.9}$	1.69	101.8	-2299.1
Tricalcium aluminate	C_3A	3.03	89.1	-3587.8
Dicalcium aluminosilicate	C_2AS	3.05	90.	
Calcium aluminodisilicate	CAS_2	2.77	100.62	

The new version of the model allows for the conversion of primary $C–S–H$ to pozzolanic $C–S–H$, for the case where there is an excess of silica due to the presence of fly ash or silica fume. The user can select to have this reaction included with or eliminated from those executed in the model through the use of a global input flag. This reaction allows for the observed reduction in the C/S ratio of $C–S–H$ over time in systems containing pozzolans [8]. When activated, the global parameter **PCSH2CSH** controls the probability of a primary $C–S–H$ pixel being converted to pozzolanic $C–S–H$ during any given dissolution cycle of the model. The calcium liberated by this reaction is converted to diffusing CH species which can form solid CH or participate in further pozzolanic reactions with the available silica and aluminosilicates.

For the purposes of incorporating fly ash, four new diffusing species [1] have been introduced into the model: diffusing anhydrite, diffusing $CaCl_2$, diffusing AS, and diffusing CAS2. The silica itself does not dissolve, but diffusing CH species react at silica surfaces to form pozzolanic $C–S–H$. The interactions between the new diffusing species and the phases of the Portland cement are as follows:

diffusing $CaCl_2$: when a diffusing $CaCl_2$ species collides with either solid or diffusing C_3A, Friedel's salt is formed. If it collides with solid C_4AF, Friedel's salt, CH, and FH_3 are formed.

diffusing AS: when a diffusing AS species collides with either solid or diffusing CH, stratlingite is formed.

Pozzolanic C–S–H Formation

$$1.1CH + S + 2.8H \rightarrow C_{1.1}SH_{3.9}$$
$$1.349 \quad 1 \quad 1.867 \quad 3.771$$

$$C_{1.7}SH_4 + 0.5H \rightarrow C_{1.1}SH_{3.9} + 0.6CH$$
$$1 \quad 0.083 \quad 0.943 \quad 0.184$$

Calcium Chloride-Aluminate Reactions

$$C_3A + CaCl_2 + 10H \rightarrow C_3A(CaCl_2)H_{10}$$
$$1.7261 \quad 1 \quad 3.487 \quad 5.747$$

$$C_4AF + CaCl_2 + 14H \rightarrow C_3A(CaCl_2)H_{10} + CH + FH_3$$
$$2.4797 \quad 1 \quad 4.882 \quad 5.747 \quad 0.641 \quad 1.3522$$

Anhydrite to Gypsum Conversion

$$C\bar{S} + 2H \rightarrow C\bar{S}H_2$$
$$1 \quad 0.69 \quad 1.423$$

Stratlingite Formation

$$2CH + AS + 6H \rightarrow C_2ASH_8$$
$$1 \quad 0.7538 \quad 1.631 \quad 3.26$$

$$CAS_2 + C_3A + 16H \rightarrow 2C_2ASH_8$$
$$1 \quad 0.886 \quad 2.862 \quad 4.286$$

$$CAS_2 + C_4AF + 20H \rightarrow 2C_2ASH_8 + CH + FH_3$$
$$0.786 \quad 1 \quad 2.81 \quad 3.37 \quad 0.2586 \quad 0.5453$$

Figure 1: Cement-fly ash model reactions - numbers below reactions indicate volume stoichiometries.

diffusing CAS2: when a diffusing CAS2 species collides with solid or diffusing C_3A, stratlingite is formed. If it collides with solid C_4AF, stratlingite, CH, and FH_3 are formed.

diffusing anhydrite: when a diffusing anhydrite species collides with either solid or diffusing gypsum, it is converted into solid gypsum.

Globally, when the model is executed under adiabatic conditions, the activation energy for the fly ash reactions (considered to be on the order of 80 kJ/mole), is used to update both the probability of a diffusing CH species reacting at a silica surface and the probability of dissolution of the aluminosilicate and calcium aluminosilicate components of the fly ash. The reactivity of the fly ash should perhaps also be a function of pH, but this has yet to be incorporated into the model. Perhaps using an exponential function to describe the pH evolution over time or as a function of degree of hydration, such as:

$$pH = pH_{init} + \gamma \times [1 - \exp(\frac{-\alpha}{1-\alpha})] \tag{1}$$

would provide a basis for estimating pH during the course of the hydration. In the above equation, pH_{init} is the initial pH of the pore solution when the cement and fly ash are mixed with water, γ is the increase in pH during the entire course of hydration, and α is the degree of hydration. The reactivity of the fly ash could then be adjusted based on the current pH of the pore solution.

The new version of the model can be found in the programs **disrealnew.c** and **hydrealnew.c** in the anonymous ftp directory as described in section 1. An example annotated input datafile for the execution of **disrealnew** is given in Appendix C.

5 Comparison to Experimental Adiabatic Heat Signature Data

For one of the French fly ashes, an extensive experimental data set of adiabatic heat signatures is available. For a variety of water-to-cement (w/c) ratios by mass, and fly ash contents, concretes have been prepared and characterized by monitoring their temperature rise over time when hydrating under adiabatic conditions. This data set can be used for a direct comparison with model predictions. This fly ash is a relatively simple one, in that it can be approximated as containing only three components: silica (S), an aluminosilicate (AS), and inert material. In addition, from SEM and image analysis, it appears that for this fly ash, most of the particles (unlike typical cement particles) are monophase. Thus, we can use the program **distfapart.c** in Appendix B to distribute the fly ash phases randomly amongst monophase particles to achieve the appropriate volume fractions (determined to be 0.33 for AS, 0.39 for S, and 0.28 for inert material based on the oxide composition determined for the fly ash). Computationally, this approach is much easier than using the detailed autocorrelation analysis to distribute the phases, as is typically performed for a cement [1].

The dissolution probability of the aluminosilicate and the reaction probability of the silica (parameter **PPOZZ**) were adjusted to provide a "best" fit to the experimental adiabatic heat signature data at w/c=0.65 and 50% fly ash content. These values were then held constant in simulations for other w/c ratios and fly ash content combinations. A heat of hydration value

of 800 J/g AS reacted was used in the model for the aluminosilicate to stratlingite conversion. This value was determined based on experiments in which the fly ash alone was hydrated in a solution of lime (CH) and the system temperature rise monitored. An activation energy of 83.14 kJ/mole (the same as that determined for the pozzolanic reaction of silica fume [9, 10]) was used in the model for all of the reactions involving one or more components of the fly ash. Example comparisons of the experimental and model temperature rise vs. time curves are provided in Figures 2, 3, and 4. In general, the agreement between model and experiment is always within a few degrees Celsius, as had been observed previously for systems with and without silica fume additions [10]. The higher model temperatures at longer times are perhaps due to more water being incorporated into the cement-fly ash hydration products than that indicated by the stoichiometries in Figure 1. Since the hydration is being executed under sealed curing conditions, it will effectively terminate when all remaining porosity is empty (as opposed to water-filled) due to chemical shrinkage. Another possibility is that certain hydration products (e.g., ettringite) are unstable at higher temperatures. If their deterioration reactions are endothermic in nature, the predicted temperature rise based on the computer model would exceed that observed experimentally. Because this fly ash is rather simple in structure and contains only a few distinct phases, further validation of the 3-D microstructural model with other fly ashes will definitely still be needed, as outlined in the next section of this report.

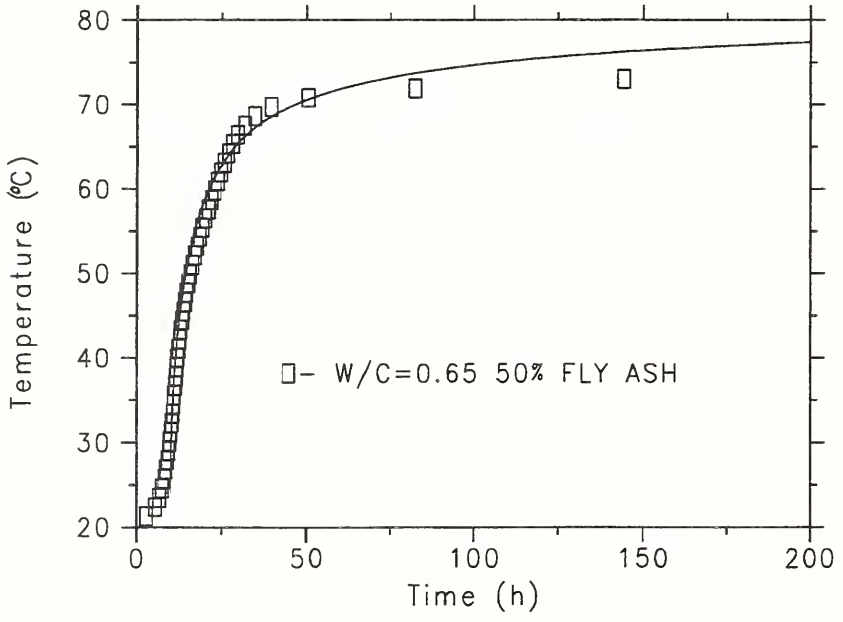


Figure 2: Comparison of experimental (data points) and calibrated simulated (solid lines) adiabatic heat signature curves for w/c=0.65 concrete with 50% fly ash content.

6 Needed Research

The best validation of the 3-D microstructural model incorporating fly ash would be a comparison of the evolution of the model volumetric phase fractions over time with the equivalent experimental data. For crystalline phases, quantitative X-ray diffraction [11] would appear to be the most straightforward means for obtaining phase quantification. This would be

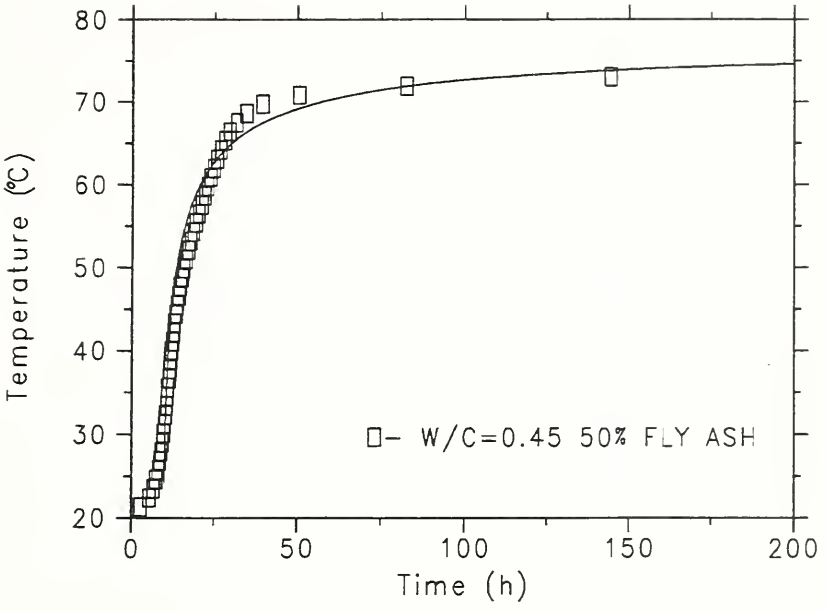


Figure 3: Comparison of experimental (data points) and simulated (solid lines) adiabatic heat signature curves for w/c=0.45 concrete with 50% fly ash content.

particularly useful for the fly ash hydration products such as Freidel's salt and stratlingite. Alternately, SEM/image analysis may be useful in analyzing real microstructures at various hydration ages. Thermogravimetric analysis provides data on the water released at different temperatures, from which one can quantitatively estimate the CH content of cement/fly ash pastes. For example, this technique has been successfully used to determine the appropriate phase and volume stoichiometries for the reaction of silica fume with cement [12].

Ideally, these experimental studies should be conducted under isothermal conditions at several temperatures. This would enable the determination of activation energies and the variation of reaction products with temperature. These studies could then be extended to hydration under adiabatic conditions as presented in the previous section. If the phases forming at high temperatures are significantly different from those at lower temperatures, extensive modifications to the computer codes may be required. For example, ettringite is known to be unstable at temperatures above 70 °C [3], so that the model could be modified to make ettringite soluble whenever the system temperature exceeds 70 °C, in addition to the current criteria for solubility based on the fraction of the initial gypsum remaining in the system.

7 Acknowledgements

The authors would like to thank Paul Stutzman of BFRL for assistance with the scanning electron microscopy, V. Waller and F. De Larrard of Laboratoire Central des Ponts et Chaussees for supplying experimental adiabatic heat signature data and material samples, and the Army Waterways Experiment Station for supplying materials and partially funding this research. SR would also like to thank the Centre Scientifique et Technique du Bâtiment and the Agence de l'Environnement et de la Maîtrise de l'Energie for partially funding this research.

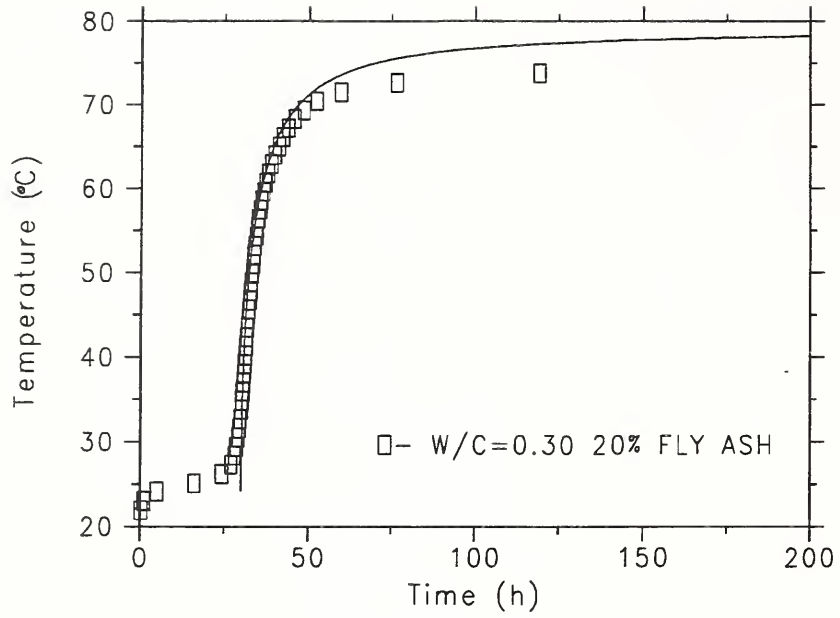


Figure 4: Comparison of experimental (data points) and simulated (solid lines) adiabatic heat signature curves for w/c=0.30 concrete with 20% fly ash content.

8 References

[1] Bentz, D.P., "Guide to Using CEMHYD3D: A Three-Dimensional Cement Hydration and Microstructure Development Modelling Package," NISTIR **5977**, U.S. Department of Commerce, February 1997.

[2] Pietersen, H.S., "Reactivity of Fly Ash and Slag in Cement," Ph.D. Thesis, Delft University of Technology, Delft, The Netherlands, 1993.

[3] Taylor, H.F.W., Cement Chemistry (Academic Press, London, 1990).

[4] Dudas, M.J., and Warren, C.J., "Submicroscopic Model of Fly Ash Particles," *Geoderma*, **40**, 101-114, 1987.

[5] Bentz, D.P., and Stutzman, P.E., "SEM Analysis and Computer Modelling of Hydration of Portland Cement Particles," in Petrography of Cementitious Materials, edited by S.M. DeHayes and D. Stark (ASTM, Philadelphia, PA, 1994) p. 60.

[6] Bentz, D.P., *A Three-Dimensional Cement Hydration and Microstructure Program. I. Hydration Rate, Heat of Hydration, and Chemical Shrinkage*, NISTIR **5756**, U.S. Department of Commerce, November 1995.

[7] Handbook of Chemistry and Physics, 63rd. ed. (CRC Press, Boca Raton, FL, 1982).

[8] Lu, P., Sun, G.K., and Young, J.F., *Journal of the American Ceramic Society*, **76**, 1003-1007, 1993.

[9] Jensen, O.M., "The pozzolanic reaction of silica fume" (in Danish) TR 229.90, Building Materials Laboratory, Technical University of Denmark, 1990.

[10] Bentz, D.P., Waller, V., and De Larrard, F., "Prediction of Adiabatic Temperature Rise in Conventional and High-Performance Concretes Using a 3-D Microstructural Model," submitted to *Cement and Concrete Research.*

[11] Stutzman, P.E., "Guide for X-Ray Powder Diffraction Analysis of Portland Cement and Clinker," NISTIR **5755**, U.S. Department of Commerce, March 1996.

[12] Atlassi, E.H., "A Quantitative Thermogravimetric Study on the Nonevaporable Water in Mature Silica Fume Concrete," Ph. D. Thesis, Chalmers University of Technology, Goteborg, August 1993.

9 Appendix A- Images of Fly Ashes

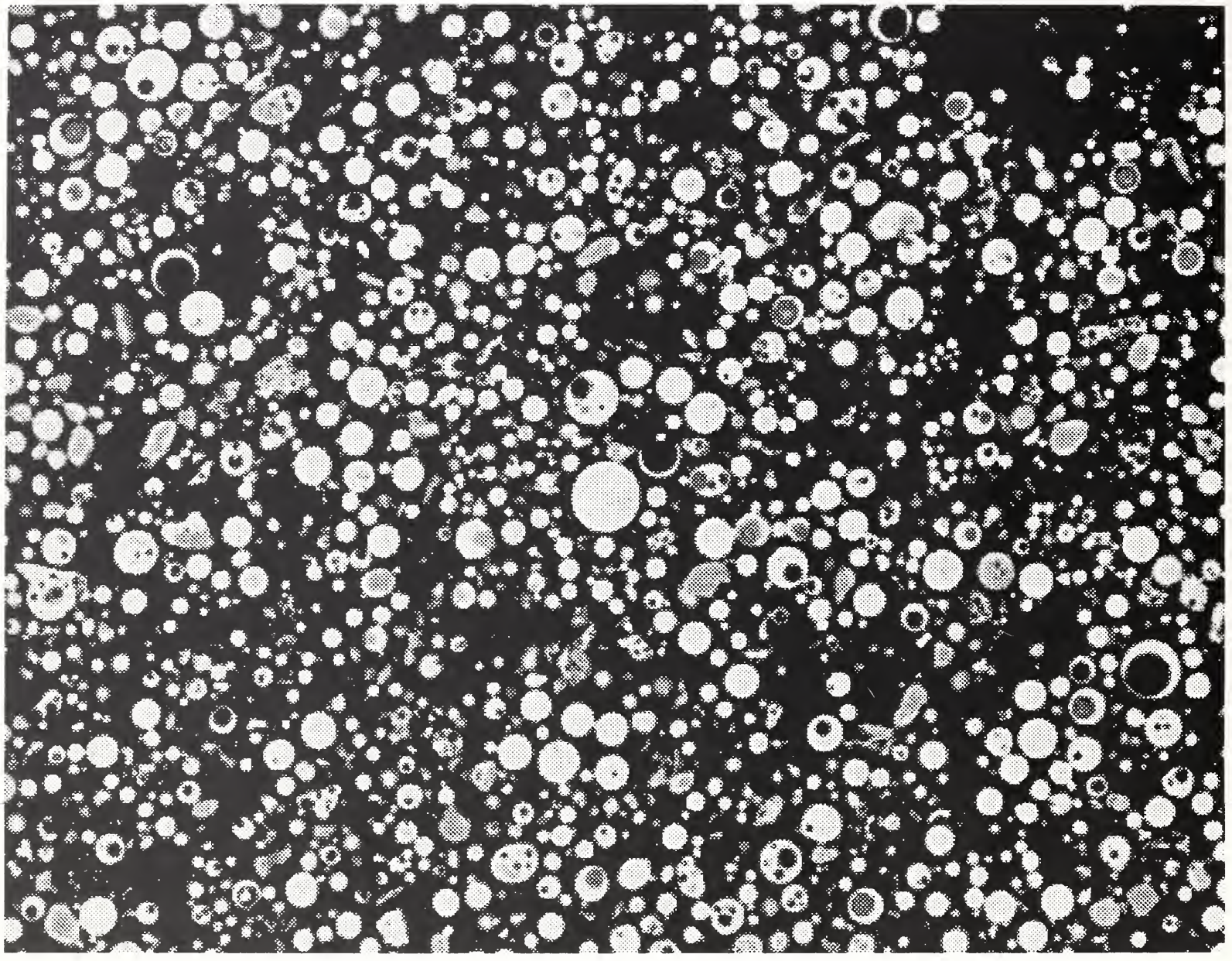


Figure 5: Initial BSE two-dimensional image of French fly ash. Image is approximately 250 μm $\times$ 200 μm.

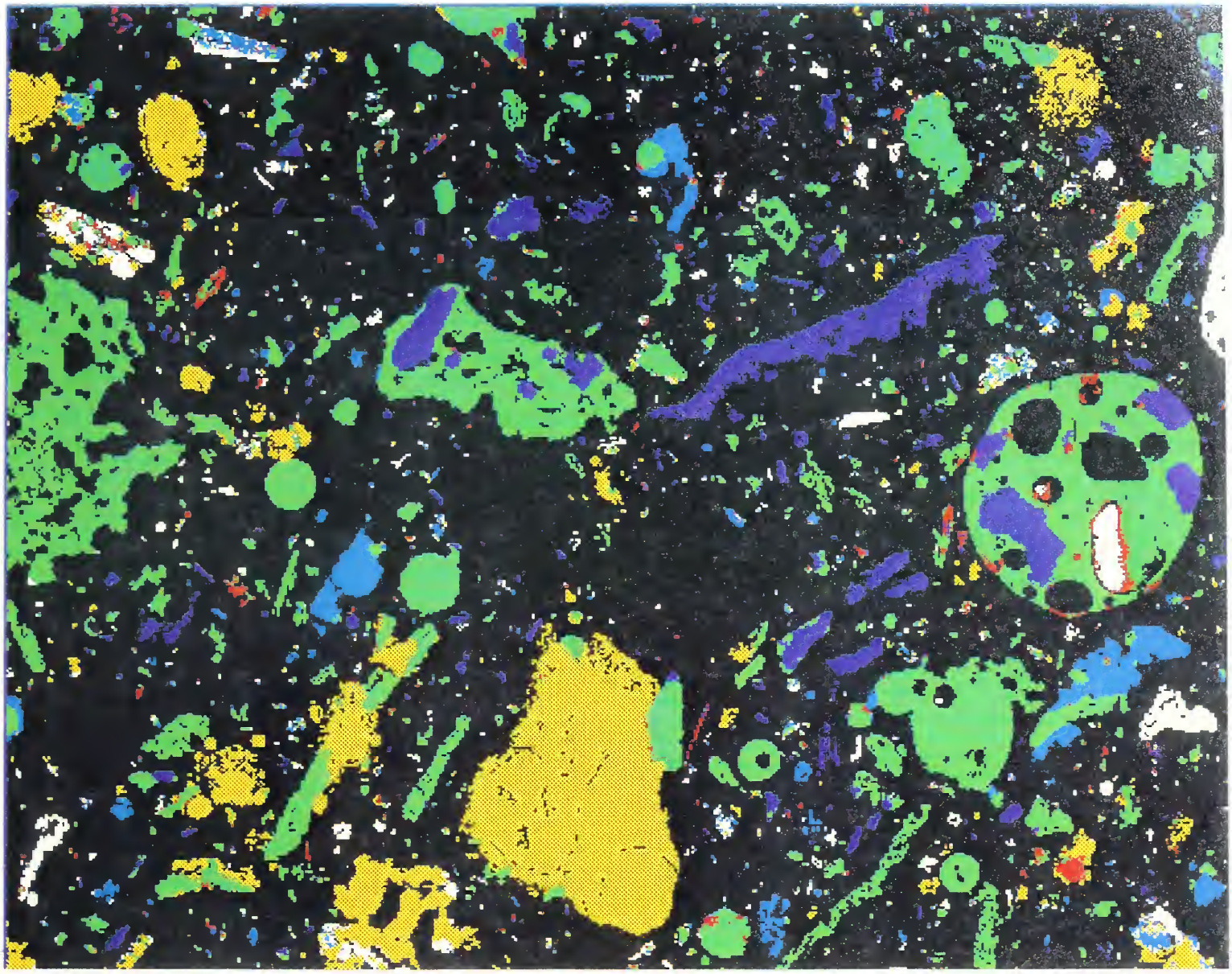


Figure 6: Final segmented two-dimensional image of French Municipal Solid Waste Incineration fly ash. Phases are color-coded as: red, S; blue, AS; green, CAS_2; orange. $CaCl_2$; aqua, anhydrite; and white, inert. Image is approximately 500 μm $\times$ 400 μm.

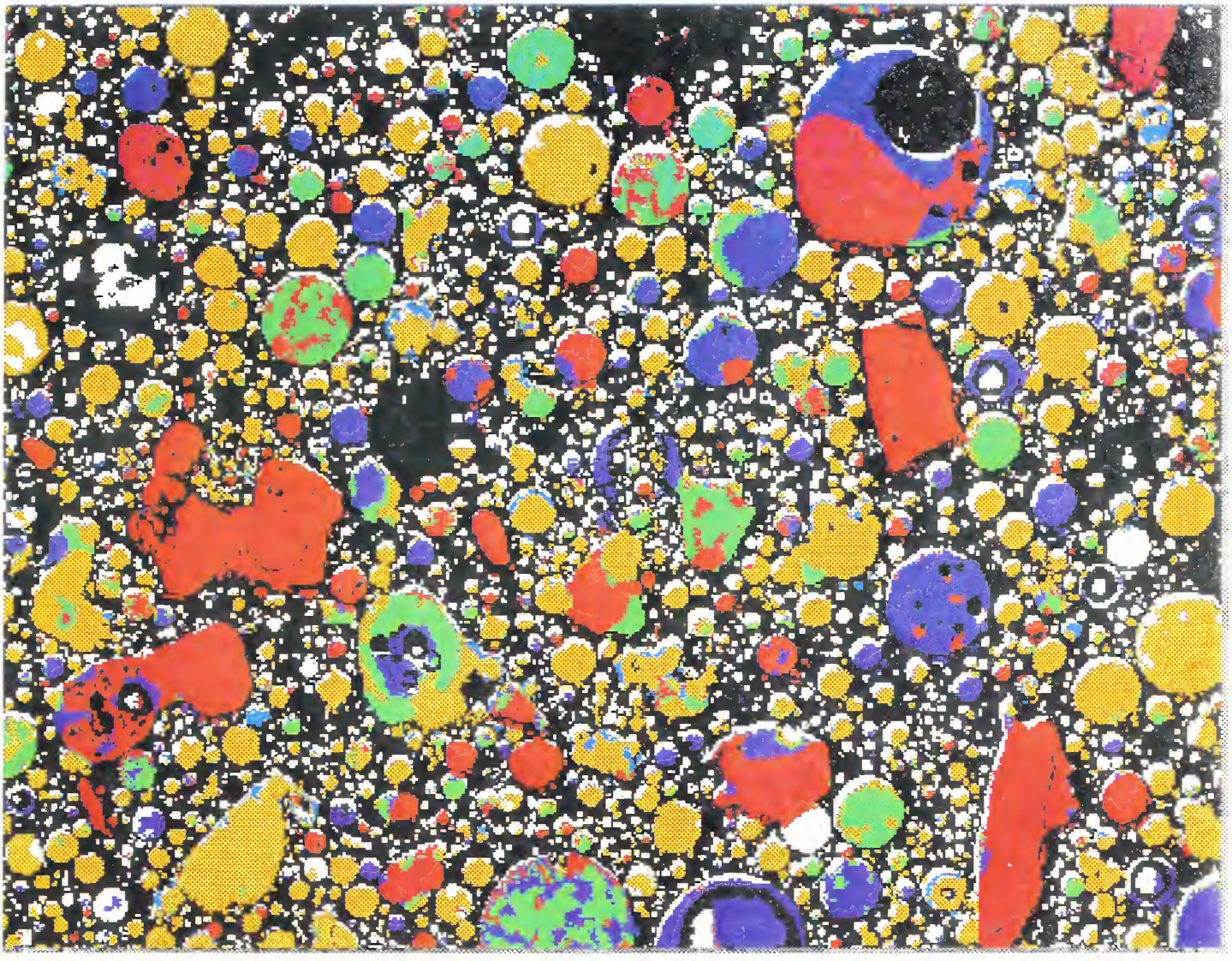


Figure 7: Final segmented two-dimensional image of Class C fly ash containing C_3A. Phases are color-coded as: red. S; blue, AS; green, CAS_2; orange. C_3A; aqua, anhydrite: and white. inert. Image is approximately 250 μm $\times$ 200 μm.

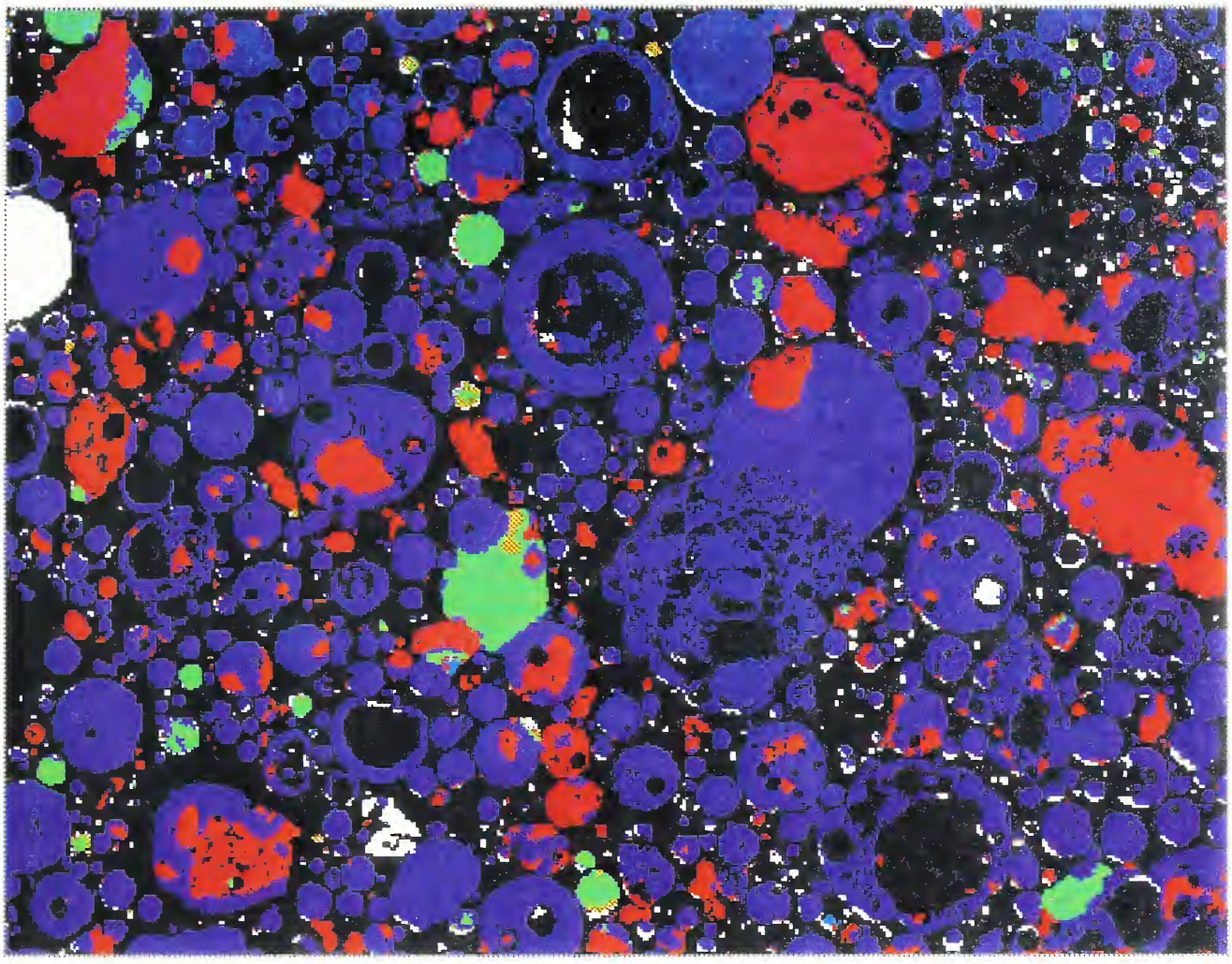


Figure 8: Final segmented two-dimensional image of Class F fly ash. Phases are color-coded as: red, S; blue, AS; green, CAS_2; orange, C_3A; aqua, anhydrite; and white, inert. Image is approximately 250 μm $\times$ 200 μm.

10 Appendix B- Computer Code Listings

10.1 ran1.c- Random number generation

```
/* Random number generator ran1 from Computers in Physics */
/* Volume 6 No. 5, 1992, 522-524, Press and Teukolsky */
/* To generate real random numbers 0.0-1.0 */
/* Should be seeded with a negative integer */
#define IA 16807
#define IM 2147483647
#define IQ 127773
#define IR 2836
#define NTAB 32
#define EPS (1.2E-07)
#define MAX(a,b) (a>b)?a:b
#define MIN(a,b) (a<b)?a:b

double ran1(idum)
int *idum;
{
        int j,k;
static int iv[NTAB],iy=0;
void nrerror();
        static double NDIV = 1.0/(1.0+(IM-1.0)/NTAB);
        static double RNMX = (1.0-EPS);
        static double AM = (1.0/IM);

if ((*idum <= 0) || (iy == 0)) {
*idum = MAX(-*idum,*idum);
                for(j=NTAB+7;j>=0;j--) {
k = *idum/IQ;
*idum = IA*(*idum-k*IQ)-IR*k;
if(*idum < 0) *idum += IM;
if(j < NTAB) iv[j] = *idum;
}
iy = iv[0];
}
k = *idum/IQ;
*idum = IA*(*idum-k*IQ)-IR*k;
if(*idum<0) *idum += IM;
j = iy*NDIV;
iy = iv[j];
iv[j] = *idum;
return MIN(AM*iy,RNMX);
}
```

```
#undef IA
#undef IM
#undef IQ
#undef IR
#undef NTAB
#undef EPS
#undef MAX
#undef MIN
```

10.2 genpart3d.c- 3-D particle generation

```
/*************************************************************************/
/*                                                                       */
/*      Program genpart3d.c to generate three-dimensional cement         */
/*              particles in a 3-D box with periodic boundaries.         */
/*      Particles are composed of either cement clinker or gypsum,       */
/*              follow a user-specified size distribution, and can       */
/*              be either flocculated, random, or dispersed.             */
/*      Programmer: Dale P. Bentz                                        */
/*                  Building and Fire Research Laboratory                */
/*                  NIST                                                 */
/*                  Building 226 Room B-350                              */
/*                  Gaithersburg, MD  20899   USA                        */
/*                  (301) 975-5865      FAX: 301-990-6891                */
/*                  E-mail: dale.bentz@nist.gov                          */
/*                                                                       */
/*************************************************************************/
/* Modified 3/97 to allow placement of pozzolanic, inert and fly ash particles*/
#include <stdio.h>
#include <math.h>

#define SYSSIZE 100     /* system size in pixels per dimension */
#define MAXTRIES 15000 /* maximum number of random tries for sphere placement */

/* phase identifiers */
#define POROSITY 0
/* Note that each particle must have a separate ID to allow for flocculation */
#define CEM 100         /* and greater */
#define CEMID 1         /* phase identifier for cement */
#define GYPID 5         /* phase identifier for gypsum */
#define AGG 6           /* phase identifier for flat aggregate */
#define POZZID 17       /* phase identifier for pozzolanic material */
#define INERTID 18      /* phase identifier for inert material */
#define FLYASH 25       /* phase identifier for generic fly ash */
#define NPARTC 12000    /* maximum number of particles allowed in box*/
#define BURNT 14000     /* this value must be at least 100 > NPARTC */
#define NUMSIZES 30     /* maximum number of different particle sizes */

/* data structure for clusters to be used in flocculation */
struct cluster{
        int partid; /* index for particle */
        int clustid;   /* ID for cluster to which this particle belongs */
        int partphase;  /* phase identifier for this particle (CEMID or GYPID)*/
```

```
        int x,y,z,r; /* particle centroid and radius in pixels */
        struct cluster *nextpart;  /* pointer to next particle in cluster */
};

/* 3-D particle structure (each particle has own ID) stored in array cement */
/* 3-D microstructure is stored in 3-D array cemreal */
static unsigned short int cement [SYSSIZE+1] [SYSSIZE+1] [SYSSIZE+1];
static unsigned short int cemreal [SYSSIZE+1] [SYSSIZE+1] [SYSSIZE+1];
int npart,aggsize;     /* global number of particles and size of aggregate */
int *seed;      /* random number seed- global */
int dispdist;   /* dispersion distance in pixels */
int clusleft;   /* number of clusters in system */
float probgyp; /* probability of gypsum particle instead of cement */
struct cluster *clust[NPARTC]; /* limit of NPARTC particles/clusters */

/* Random number generator ran1 from Computers in Physics */
/* Volume 6 No. 5, 1992, 522-524, Press and Teukolsky */
/* To generate real random numbers 0.0-1.0 */
/* Should be seeded with a negative integer */
#define IA 16807
#define IM 2147483647
#define IQ 127773
#define IR 2836
#define NTAB 32
#define EPS (1.2E-07)
#define MAX(a,b) (a>b)?a:b
#define MIN(a,b) (a<b)?a:b

double ran1(idum)
int *idum;
/* Calls: no routines */
/* Called by: gsphere,makefloc */
{
        int j,k;
        static int iv[NTAB],iy=0;
        void nrerror();
        static double NDIV = 1.0/(1.0+(IM-1.0)/NTAB);
        static double RNMX = (1.0-EPS);
        static double AM = (1.0/IM);

        if ((*idum <= 0) || (iy == 0)) {
                *idum = MAX(-*idum,*idum);
                for(j=NTAB+7;j>=0;j--) {
                        k = *idum/IQ;
                        *idum = IA*(*idum-k*IQ)-IR*k;
                        if(*idum < 0) *idum += IM;
```

```
                        if(j < NTAB) iv[j] = *idum;
                }
                iy = iv[0];
        }
        k = *idum/IQ;
        *idum = IA*(*idum-k*IQ)-IR*k;
        if(*idum<0) *idum += IM;
        j = iy*NDIV;
        iy = iv[j];
        iv[j] = *idum;
        return MIN(AM*iy,RNMX);
}
#undef IA
#undef IM
#undef IQ
#undef IR
#undef NTAB
#undef EPS
#undef MAX
#undef MIN

/* routine to add a flat plate aggregate in the microstructure */
void addagg()
/* Calls: no other routines */
/* Called by: main program */
{
        int ix,iy,iz;
        int agglo,agghi;

/* Be sure aggregate size is an even integer */
        do{
             printf("Enter thickness of aggregate to place (even integer) \n");
             scanf("%d",&aggsize);
             printf("%d\n",aggsize);
        } while ((aggsize%2)!=0);

        if(aggsize!=0){
        agglo=(SYSSIZE/2)-((aggsize-2)/2);
        agghi=(SYSSIZE/2)+(aggsize/2);

        /* Aggregate is placed in yz plane */
        for(ix=agglo;ix<=agghi;ix++){
        for(iy=1;iy<=SYSSIZE;iy++){
        for(iz=1;iz<=SYSSIZE;iz++){

        /* Mark aggregate into both particle and microstructure images */
```

```
                cement [ix][iy][iz]=AGG;
                cemreal [ix][iy][iz]=AGG;
        }
        }
        }
        }
}

/* routine to check or perform placement of sphere of ID phasein */
/* centered at location (xin,yin,zin) of radius radd */
/* wflg=1 check for fit of sphere */
/* wflg=2 place the sphere */
/* phasein and phase2 are phases to assign to cement and cemreal images resp. */
int chksph(xin,yin,zin,radd,wflg,phasein,phase2)
        int xin,yin,zin,radd,wflg,phasein,phase2;
/* Calls: no other routines */
/* Called by: gsphere */
{
        int nofits,xp,yp,zp,i,j,k;
        float dist,xdist,ydist,zdist,ftmp;

        nofits=0;       /* Flag indicating if placement is possible */

/* Check all pixels within the digitized sphere volume */
        for(i=xin-radd;((i<=xin+radd)&&(nofits==0));i++){
                xp=i;
                /* use periodic boundary conditions for sphere placement */
                if(xp<1) {xp+=SYSSIZE;}
                else if(xp>SYSSIZE) {xp-=SYSSIZE;}
                ftmp=(float)(i-xin);
                xdist=ftmp*ftmp;
        for(j=yin-radd;((j<=yin+radd)&&(nofits==0));j++){
                yp=j;
                /* use periodic boundary conditions for sphere placement */
                if(yp<1) {yp+=SYSSIZE;}
                else if(yp>SYSSIZE) {yp-=SYSSIZE;}
                ftmp=(float)(j-yin);
                ydist=ftmp*ftmp;
        for(k=zin-radd;((k<=zin+radd)&&(nofits==0));k++){
                zp=k;
                /* use periodic boundary conditions for sphere placement */
                if(zp<1) {zp+=SYSSIZE;}
                else if(zp>SYSSIZE) {zp-=SYSSIZE;}
                ftmp=(float)(k-zin);
                zdist=ftmp*ftmp;
```

```
        /* Compute distance from center of sphere to this pixel */
                dist=sqrt(xdist+ydist+zdist);
                if((dist-0.5)<=(float)radd){
                        /* Perform placement */
                        if(wflg==2){
                                cement [xp] [yp] [zp]=phasein;
                                cemreal [xp][yp][zp]=phase2;
                        }
                        /* or check placement */
                        else if((wflg==1)&&(cement [xp] [yp] [zp] !=POROSITY)){
                                nofits=1;
                        }
                }
                /* Check for overlap with aggregate */
      if((wflg==1)&&((abs(xp-((float)(SYSSIZE+1)/2.0)))<((float)aggsize/2.0))){
                        nofits=1;
                }
        }
        }
        }

        /* return flag indicating if sphere will fit */
        return(nofits);
}

/* routine to place spheres of various sizes and phases at random */
/* locations in 3-D microstructure */
/* numgen is number of different size spheres to place */
/* numeach holds the number of each size class */
/* sizeeach holds the radius of each size class */
/* pheach holds the phase of each size class */
void gsphere(numgen,numeach,sizeeach,pheach)
        int numgen;
        long int numeach[NUMSIZES];
        int sizeeach[NUMSIZES],pheach[NUMSIZES];
/* Calls: makesph, ran1 */
/* Called by: create */
{
        int count,x,y,z,radius,ig,tries,phnow;
        long int jg;
        float rx,ry,rz,testgyp;
        struct cluster *partnew;

/* Generate spheres of each size class in turn (largest first) */
        for(ig=0;ig<numgen;ig++){
```

```
phnow=pheach[ig];        /* phase for this class */
                radius=sizeeach[ig];     /* radius for this class */
                /* loop for each sphere in this size class */
                for(jg=1;jg<=numeach[ig];jg++){

                        tries=0;
                        /* Stop after MAXTRIES random tries */
                        do{
                                tries+=1;
                        /* generate a random center location for the sphere */
                                x=(int)((float)SYSSIZE*ran1(seed))+1;
                                y=(int)((float)SYSSIZE*ran1(seed))+1;
                                z=(int)((float)SYSSIZE*ran1(seed))+1;
                                /* See if the sphere will fit at x,y,z */
                                /* Include dispersion distance when checking */
                            /* to insure requested separation between spheres */
                              count=chksph(x,y,z,radius+dispdist,1,npart+CEM,0);
                                if(tries>MAXTRIES){
printf("Could not place sphere %d after %d random attempts \n",npart,MAXTRIES);
                                        exit(1);
                                }
                        } while(count!=0);

                        /* place the sphere at x,y,z */
                        npart+=1;
                        if(npart>=NPARTC){
                                printf("Too many spheres being generated \n");
            printf("User needs to increase value of NPARTC at top of C-code\n");
                                exit(1);
                        }
                        /* Allocate space for new particle info */
                  clust[npart]=(struct cluster *)malloc(sizeof(struct cluster));
                        clust[npart]->partid=npart;
                        clust[npart]->clustid=npart;
                        /* Default to cement placement */
                        clust[npart]->partphase=CEMID;
                        clust[npart]->x=x;
                        clust[npart]->y=y;
                        clust[npart]->z=z;
                        clust[npart]->r=radius;
                        clusleft+=1;
if(phnow==1){
                        testgyp=ran1(seed);
                  /* Do not use dispersion distance when placing particle */
                        if(testgyp>probgyp){
                                count=chksph(x,y,z,radius,2,npart+CEM-1,CEMID);
```

```
                        }
                        else{
                                /* Place particle as gypsum */
                                count=chksph(x,y,z,radius,2,npart+CEM-1,GYPID);
                                /* Correct phase ID of particle */
                                clust[npart]->partphase=GYPID;
                        }
}
/* place as inert or pozzolanic material */
else{
count=chksph(x,y,z,radius,2,npart+CEM-1,phnow);
                                /* Correct phase ID of particle */
                                clust[npart]->partphase=phnow;
}
                        clust[npart]->nextpart=NULL;
                }
        }
}

/* routine to obtain user input and create a starting microstructure */
void create()
/* Calls: gsphere */
/* Called by: main program */
{
        int numsize,sphrad [NUMSIZES],sphase[NUMSIZES];
        long int sphnum [NUMSIZES],inval1;
        int isph,inval;

        do{
printf("Enter number of different size spheres to use(max. is %d) \n",NUMSIZES);
                scanf("%d",&numsize);
                printf("%d \n",numsize);
        }while ((numsize>NUMSIZES)||(numsize<0));
        do{
                printf("Enter dispersion factor for spheres (0-2) \n");
                scanf("%d",&dispdist);
                printf("%d \n",dispdist);
        }while ((dispdist<0)||(dispdist>2));
        do{
                printf("Enter probability for gypsum particles (0.0-1.0) \n");
                scanf("%f",&probgyp);
                printf("%f \n",probgyp);
        } while ((probgyp<0.0)||(probgyp>1.0));

        if((numsize>0)&&(numsize<(NUMSIZES+1))){
       printf("Enter number, radius, and phase for each class (largest radius 1st) \n"
```

```
printf("Phases are 1- Cement and gypsum, 17- Pozzolanic, 18- Inert 25- Fly Ash \n")

        /* Obtain input for each size class of spheres */
                for(isph=0;isph<numsize;isph++){
                        printf("Enter number of spheres of class %d \n",isph+1);
                        scanf("%ld",&inval1);
                        printf("%ld \n",inval1);
                        sphnum[isph]=inval1;
                        do{
                        printf("Enter radius of spheres of class %d \n",isph+1);
                                printf("(Integer <=%d please) \n",SYSSIZE/5);
                                scanf("%d",&inval);
                                printf("%d \n",inval);
                        } while ((inval<1)||(inval>(SYSSIZE/5)));
                        sphrad[isph]=inval;
                        do{
                        printf("Enter phase of spheres of class %d \n",isph+1);
                                scanf("%d",&inval);
                                printf("%d \n",inval);
                        } while ((inval!=1)&&(inval!=12)&&(inval!=13)&&(inval!=25))
                        sphase[isph]=inval;

                }
                gsphere(numsize,sphnum,sphrad,sphase);
        }
}

/* Routine to draw a particle during flocculation routine */
/* See routine chksph for definition of parameters */
void drawfloc(xin,yin,zin,radd,phasein,phase2)
        int xin,yin,zin,radd,phasein,phase2;
/* Calls: no other routines */
/* Called by: makefloc */
{
        int xp,yp,zp,i,j,k;
        float dist,xdist,ydist,zdist,ftmp;

/* Check all pixels within the digitized sphere volume */
        for(i=xin-radd;(i<=xin+radd);i++){
                xp=i;
                /* use periodic boundary conditions for sphere placement */
                if(xp<1) {xp+=SYSSIZE;}
                else if(xp>SYSSIZE) {xp-=SYSSIZE;}
                ftmp=(float)(i-xin);
                xdist=ftmp*ftmp;
        for(j=yin-radd;(j<=yin+radd);j++){
```

```
                yp=j;
                /* use periodic boundary conditions for sphere placement */
                if(yp<1) {yp+=SYSSIZE;}
                else if(yp>SYSSIZE) {yp-=SYSSIZE;}
                ftmp=(float)(j-yin);
                ydist=ftmp*ftmp;
        for(k=zin-radd;(k<=zin+radd);k++){
                zp=k;
                /* use periodic boundary conditions for sphere placement */
                if(zp<1) {zp+=SYSSIZE;}
                else if(zp>SYSSIZE) {zp-=SYSSIZE;}
                ftmp=(float)(k-zin);
                zdist=ftmp*ftmp;

        /* Compute distance from center of sphere to this pixel */
                dist=sqrt(xdist+ydist+zdist);
                if((dist-0.5)<=(float)radd){
                        /* Update both cement and cemreal images */
                        cement [xp] [yp] [zp]=phasein;
                        cemreal [xp] [yp] [zp]=phase2;
                }
        }
        }
        }
}

/* Routine to check particle placement during flocculation */
/* for particle of size radd centered at (xin,yin,zin) */
/* Returns flag indicating if placement is possible */
int chkfloc(xin,yin,zin,radd)
        int xin,yin,zin,radd;
/* Calls: no other routines */
/* Called by: makefloc */
{
        int nofits,xp,yp,zp,i,j,k;
        float dist,xdist,ydist,zdist,ftmp;

        nofits=0;       /* Flag indicating if placement is possible */

/* Check all pixels within the digitized sphere volume */
        for(i=xin-radd;((i<=xin+radd)&&(nofits==0));i++){
                xp=i;
                /* use periodic boundary conditions for sphere placement */
                if(xp<1) {xp+=SYSSIZE;}
                else if(xp>SYSSIZE) {xp-=SYSSIZE;}
                ftmp=(float)(i-xin);
```

```
                xdist=ftmp*ftmp;
        for(j=yin-radd;((j<=yin+radd)&&(nofits==0));j++){
                yp=j;
                /* use periodic boundary conditions for sphere placement */
                if(yp<1) {yp+=SYSSIZE;}
                else if(yp>SYSSIZE) {yp-=SYSSIZE;}
                ftmp=(float)(j-yin);
                ydist=ftmp*ftmp;
        for(k=zin-radd;((k<=zin+radd)&&(nofits==0));k++){
                zp=k;
                /* use periodic boundary conditions for sphere placement */
                if(zp<1) {zp+=SYSSIZE;}
                else if(zp>SYSSIZE) {zp-=SYSSIZE;}
                ftmp=(float)(k-zin);
                zdist=ftmp*ftmp;

        /* Compute distance from center of sphere to this pixel */
                dist=sqrt(xdist+ydist+zdist);
                if((dist-0.5)<=(float)radd){
                        if((cement [xp] [yp] [zp] !=POROSITY)){
                                /* Record ID of particle hit */
                                nofits=cement [xp] [yp] [zp];
                        }
                }
                /* Check for overlap with aggregate */
                if((abs(xp-((float)(SYSSIZE+1)/2.0)))<((float)aggsize/2.0)){
                        nofits=AGG;
                }
        }
        }
        }
        /* return flag indicating if sphere will fit */
        return(nofits);
}

/* routine to perform flocculation of particles */
void makefloc()
/* Calls: drawfloc, chkfloc, ran1 */
/* Called by: main program */
{
        int partdo,numfloc;
        int nstart;
        int nleft,ckall;
        int xm,ym,zm,moveran;
        int xp,yp,zp,rp,clushit,valkeep;
        int iclus,cluspart[NPARTC];
```

```
struct cluster *parttmp,*partpoint,*partkeep;

nstart=npart; /* Counter of number of flocs remaining */
for(iclus=1;iclus<=npart;iclus++){
          cluspart[iclus]=iclus;
}
do{
  printf("Enter number of flocs desired at end of routine (>0) \n");
        scanf("%d",&numfloc);
        printf("%d\n",numfloc);
} while (numfloc<=0);

while(nstart>numfloc){
        nleft=0;

        /* Try to move each cluster in turn */
        for(iclus=1;iclus<=npart;iclus++){
                if(clust[iclus]==NULL){
                        nleft+=1;
                }
                else{
                        xm=ym=zm=0;
        /* Generate a random move in one of 6 principal directions */
        moveran=6.*ran1(seed);
        switch(moveran){
                        case 0:
                                xm=1;
                                break;
                        case 1:
                                xm=(-1);
                                break;
                        case 2:
                                ym=1;
                                break;
                        case 3:
                                ym=(-1);
                                break;
                        case 4:
                                zm=1;
                                break;
                        case 5:
                                zm=(-1);
                                break;
                        default:
                                break;
        }
```

```
/* First erase all particles in cluster */
partpoint=clust[iclus];

while(partpoint!=NULL){
        xp=partpoint->x;
        yp=partpoint->y;
        zp=partpoint->z;
        rp=partpoint->r;
        drawfloc(xp,yp,zp,rp,0,0);
        partpoint=partpoint->nextpart;
}

ckall=0;
/* Now try to draw cluster at new location */
partpoint=clust[iclus];
while((partpoint!=NULL)&&(ckall==0)){
        xp=partpoint->x+xm;
        yp=partpoint->y+ym;
        zp=partpoint->z+zm;
        rp=partpoint->r;
        ckall=chkfloc(xp,yp,zp,rp);
        partpoint=partpoint->nextpart;
}

if(ckall==0){
/* Place cluster particles at new location */
        partpoint=clust[iclus];
        while(partpoint!=NULL){
                xp=partpoint->x+xm;
                yp=partpoint->y+ym;
                zp=partpoint->z+zm;
                rp=partpoint->r;
                valkeep=partpoint->partphase;
                partdo=partpoint->partid;
                drawfloc(xp,yp,zp,rp,partdo+CEM-1,valkeep);
                /* Update particle location */
                partpoint->x=xp;
                partpoint->y=yp;
                partpoint->z=zp;
                partpoint=partpoint->nextpart;
        }
}
else{
        /* A cluster or aggregate was hit */
        /* Draw particles at old location */
        partpoint=clust[iclus];
```

```
                        /* partkeep stores pointer to last particle in list */
                        while(partpoint!=NULL){
                                xp=partpoint->x;
                                yp=partpoint->y;
                                zp=partpoint->z;
                                rp=partpoint->r;
                                valkeep=partpoint->partphase;
                                partdo=partpoint->partid;
                                drawfloc(xp,yp,zp,rp,partdo+CEM-1,valkeep);
                                partkeep=partpoint;
                                partpoint=partpoint->nextpart;
                        }
                        /* Determine the cluster hit */
                        if(ckall!=AGG){
                                clushit=cluspart[ckall-CEM+1];
      /* Move all of the particles from cluster clushit to cluster iclus */
                                parttmp=clust[clushit];
                                /* Attach new cluster to old one */
                                partkeep->nextpart=parttmp;
                                while(parttmp!=NULL){
                                        cluspart[parttmp->partid]=iclus;
                    /* Relabel all particles added to this cluster */
                                        parttmp->clustid=iclus;
                                        parttmp=parttmp->nextpart;
                                }
                                /* Disengage the cluster that was hit */
                                clust[clushit]=NULL;
                                nstart-=1;
                        }
                }
                }
        }
  printf("Number left was %d but number of clusters is %d \n",nleft,nstart);
        }       /* end of while loop */
        clusleft=nleft;
}

/* routine to assess global phase fractions present in 3-D system */
void measure()
/* Calls: no other routines */
/* Called by: main program */
{
        long int npor,ngyp,ncem,nagg,npozz,ninert,nflyash;
        int i,j,k,valph;

/* counters for the various phase fractions */
```

```
        npor=0;
        ngyp=0;
        ncem=0;
        nagg=0;
ninert=0;
npozz=0;
nflyash=0;

/* Check all pixels in 3-D microstructure */
        for(i=1;i<=SYSSIZE;i++){
        for(j=1;j<=SYSSIZE;j++){
        for(k=1;k<=SYSSIZE;k++){
                valph=cemreal [i] [j] [k];
                if(valph==POROSITY) {npor+=1;}
                else if(valph==CEMID){ncem+=1;}
                else if(valph==GYPID){ngyp+=1;}
                else if(valph==AGG) {nagg+=1;}
                else if(valph==POZZID) {npozz+=1;}
                else if(valph==INERTID) {ninert+=1;}
                else if(valph==FLYASH) {nflyash+=1;}
        }
        }
        }

/* Output results */
        printf("Porosity= %ld \n",npor);
        printf("Cement= %ld \n",ncem);
        printf("Gypsum= %ld \n",ngyp);
        printf("Aggregate= %ld \n",nagg);
printf("Pozzolan= %ld \n",npozz);
printf("Inert= %ld \n",ninert);
printf("Fly Ash= %ld \n",nflyash);
}

/* Routine to measure phase fractions as a function of distance from */
/* aggregate surface      */
void measagg()
/* Calls: no other routines */
/* Called by: main program */
{
        int phase [24],ptot;
        int icnt,ixlo,ixhi,iy,iz,phid,idist;

        printf("Distance  Porosity  Cement  Gypsum  Pozzolan  Inert   Fly Ash\n");

/* Increase distance from aggregate in increments of one */
```

```
        for(idist=1;idist<=(SYSSIZE-aggsize)/2;idist++){
                /* Pixel left of aggregate surface */
                ixlo=((SYSSIZE-aggsize+2)/2)-idist;
                /* Pixel right of aggregate surface */
                ixhi=((SYSSIZE+aggsize)/2)+idist;

        /* Initialize phase counts for this distance */
                for(icnt=0;icnt<21;icnt++){
                        phase[icnt]=0;
                }
                ptot=0;

/* Check all pixels which are this distance from aggregate surface */
                for(iy=1;iy<=SYSSIZE;iy++){
                for(iz=1;iz<=SYSSIZE;iz++){
                        phid=cemreal [ixlo] [iy] [iz];
                        ptot+=1;
                        if(phid<21){
                                phase[phid]+=1;
                        }
                        phid=cemreal [ixhi] [iy] [iz];
                        ptot+=1;
                        if(phid<21){
                                phase[phid]+=1;
                        }
                }
                }

                /* Output results for this distance from surface */
     printf("%d   %d   %d  %d  %d  %d %d\n",idist,phase[0],phase[CEMID],phase[GYPID],p

         }
}

/* routine to assess the connectivity (percolation) of a single phase */
/* Two matrices are used here: one for the current burnt locations */
/*                the other to store the newly found burnt locations */
void connect()
/* Calls: no other routines */
/* Called by: main program */
{
        long int inew,ntop,nthrough,ncur,nnew,ntot;
        int i,j,k,nmatx[29000],nmaty[29000],nmatz[29000];
     int xcn,ycn,zcn,npix,x1,y1,z1,igood,nnewx[29000],nnewy[29000],nnewz[29000];
        int jnew,icur;
```

```
        do{
                printf("Enter phase to analyze 0) pores 1) Cement  \n");
                scanf("%d",&npix);
                printf("%d \n",npix);
        } while ((npix!=0)&&(npix!=1));

        /* counters for number of pixels of phase accessible from top surface */
        /* and number which are part of a percolated pathway */
        ntop=0;
        nthrough=0;

        /* percolation is assessed from top to bottom only */
        /* and burning algorithm is nonperiodic in x and y directions */
        k=1;
        for(i=1;i<=SYSSIZE;i++){
                for(j=1;j<=SYSSIZE;j++){
                        ncur=0;
                        ntot=0;
                        igood=0;     /* Indicates if bottom has been reached */
if(((cement [i] [j] [k]==npix)&&((cement [i] [j] [SYSSIZE]==npix)||
(cement [i] [j] [SYSSIZE]==(npix+BURNT))))||((cement [i] [j] [SYSSIZE]>=CEM)&&
(cement [i] [j] [k]>=CEM)&&(cement [i] [j] [k]<BURNT)&&(npix==1))){
                                /* Start a burn front */
                                cement [i] [j] [k]+=BURNT;
                                ntot+=1;
                                ncur+=1;
                                /* burn front is stored in matrices nmat* */
                                /* and nnew* */
                                nmatx[ncur]=i;
                                nmaty[ncur]=j;
                                nmatz[ncur]=1;
                            /* Burn as long as new (fuel) pixels are found */
                                do{
                                        nnew=0;
                                        for(inew=1;inew<=ncur;inew++){
                                                xcn=nmatx[inew];
                                                ycn=nmaty[inew];
                                                zcn=nmatz[inew];

                                                /* Check all six neighbors */
                                                for(jnew=1;jnew<=6;jnew++){
                                                        x1=xcn;
                                                        y1=ycn;
                                                        z1=zcn;
                                                        if(jnew==1){
                                                                x1-=1;
```

```
                                        if(x1<1){
                                            x1+=SYSSIZE;
                                        }
                                    }
                                    else if(jnew==2){
                                        x1+=1;
                                        if(x1>SYSSIZE){
                                            x1-=SYSSIZE;
                                        }
                                    }
                                    else if(jnew==3){
                                        y1-=1;
                                        if(y1<1){
                                            y1+=SYSSIZE;
                                        }
                                    }
                                    else if(jnew==4){
                                        y1+=1;
                                        if(y1>SYSSIZE){
                                            y1-=SYSSIZE;
                                        }
                                    }
                                    else if(jnew==5){
                                        z1-=1;
                                    }
                                    else if(jnew==6){
                                        z1+=1;
                                    }

/* Nonperiodic in z direction so be sure to remain in the 3-D box */
                                    if((z1>=1)&&(z1<=SYSSIZE)){
if((cement [x1] [y1] [z1]==npix)||((cement [x1] [y1] [z1]>=CEM)&&
(cement [x1] [y1] [z1]<BURNT)&&(npix==1))){
                                        ntot+=1;
                                    cement [x1] [y1] [z1]+=BURNT;
                                        nnew+=1;
                                      if(nnew>=29000){
                                printf("error in size of nnew \n");
                                        }
                                        nnewx[nnew]=x1;
                                        nnewy[nnew]=y1;
                                        nnewz[nnew]=z1;
                            /* See if bottom of system has been reached */
                                      if(z1==SYSSIZE){
                                        igood=1;
                                      }
```

```
                                                        }
                                                }
                                                }
                                        }
                                        if(nnew>0){
                                                ncur=nnew;
                                        /* update the burn front matrices */
                                                for(icur=1;icur<=ncur;icur++){
                                                        nmatx[icur]=nnewx[icur];
                                                        nmaty[icur]=nnewy[icur];
                                                        nmatz[icur]=nnewz[icur];
                                                }
                                        }
                                }while (nnew>0);

                                ntop+=ntot;
                                if(igood==1){
                                         nthrough+=ntot;
                                }
                        }
                }
        }

        printf("Phase ID= %d \n",npix);
        printf("Number accessible from top= %ld \n",ntop);
        printf("Number contained in through pathways= %ld \n",nthrough);

        /* return the burnt sites to their original phase values */
        for(i=1;i<=SYSSIZE;i++){
        for(j=1;j<=SYSSIZE;j++){
        for(k=1;k<=SYSSIZE;k++){
                if(cement [i] [j] [k]>=BURNT){
                        cement [i] [j] [k]-=BURNT;
                }
        }
        }
        }
}

/* Routine to output final microstructure to file */
void outmic()
/* Calls: no other routines */
/* Called by: main program */
{
        FILE *outfile,*partfile;
        char filen[80],filepart[80];
```

```
        int ix,iy,iz,valout;

        printf("Enter name of file to save microstructure to \n");
        scanf("%s",filen);
        printf("%s\n",filen);

        outfile=fopen(filen,"w");

        printf("Enter name of file to save particle IDs to \n");
        scanf("%s",filepart);
        printf("%s\n",filepart);

        partfile=fopen(filepart,"w");

        for(iz=1;iz<=SYSSIZE;iz++){
        for(iy=1;iy<=SYSSIZE;iy++){
        for(ix=1;ix<=SYSSIZE;ix++){
                valout=cemreal[ix][iy][iz];
                fprintf(outfile,"%1d\n",valout);
                valout=cement[ix][iy][iz];
                if(valout<0){valout=0;}
                fprintf(partfile,"%d\n",valout);
        }
        }
        }
        fclose(outfile);
        fclose(partfile);
}

main(){
        int userc;      /* User choice from menu */
        int nseed,ig,jg,kg;

        printf("Enter random number seed value \n");
        scanf("%d",&nseed);
        printf("%d \n",nseed);
        seed=(&nseed);

        /* Initialize counters and system parameters */
        npart=0;
        aggsize=0;
        clusleft=0;

        /* clear the 3-D system to all porosity to start */
        for(ig=1;ig<=SYSSIZE;ig++){
        for(jg=1;jg<=SYSSIZE;jg++){
```

```
        for(kg=1;kg<=SYSSIZE;kg++){
                cement [ig] [jg] [kg]=POROSITY;
                cemreal [ig] [jg] [kg]=POROSITY;
        }
        }
        }

        /* present menu and execute user choice */
        do{
printf(" \n Input User Choice \n");
printf("1) Exit \n");
printf("2) Add spherical particles (cement,gypsum, pozzolans, etc.) to microstructu
printf("3) Flocculate system by reducing number of particle clusters \n");
printf("4) Measure phase fractions \n");
printf("5) Add an aggregate to the microstructure \n");
printf("6) Measure single phase connectivity (pores or solids) \n");
printf("7) Measure phase fractions vs. distance from aggregate \n");
printf("8) Output microstructure to file \n");

                scanf("%d",&userc);
                printf("%d \n",userc);
                fflush(stdout);

                switch (userc) {
                        case 2:
                                create();
                                break;
                        case 3:
                                makefloc();
                                break;
                        case 4:
                                measure();
                                break;
                        case 5:
                                addagg();
                                break;
                        case 6:
                                connect();
                                break;
                        case 7:
                                if(aggsize!=0){
                                measagg();
        }
        else{
        printf("No aggregate present. \n");
        }
```

```
                        break;
                case 8:
                        outmic();
                        break;
                default:
                        break;
            }
        } while (userc!=1);
}
```

10.3 distfarand.c- Random pixel distribution of fly ash

```
#include <stdio.h>
#include <math.h>

/*#########################################################################*/
/* Program distfarand                                                      */
/* Purpose: To distribute fly ash phases randomly amongst particles        */
/* Programmer: Dale P. Bentz                                               */
/*                                                                         */
/*#########################################################################*/

/* Define phase IDs */
#define FLYASH 25
#define C3A 35
#define CACL2 12
#define ASGID 14
#define CAS2ID 16
#define INERTID 18
#define POZZID 17
#define ANHYDRITE 20
#define SYSSIZE 100

int *seed;
#include "ran1.c"

main()
{
        FILE *infile,*outfile;
        char filein[80],fileout[80];
        int ix,iy,iz,valin,valout;
        float probasg,probcacl2,probsio2;
        float prph,probanh,probcas2,probc3a;
        int nseed;

        printf("Enter random number seed value (<0)\n");
        scanf("%d",&nseed);
        printf("%d\n",nseed);
        seed=(&nseed);

        printf("Enter name of file for input \n");
        scanf("%s",filein);
        printf("%s\n",filein);
```

```
printf("Enter name of file for output \n");
scanf("%s",fileout);
printf("%s\n",fileout);

/* Get user input for phase probabilities (volume fractions) */
printf("Enter probability for fly ash to be aluminosilicate glass \n");
scanf("%f",&probasg);
printf("%f\n",probasg);
printf("Enter probability for fly ash to be calcium aluminodisilicate \n");
scanf("%f",&probcas2);
printf("%f\n",probcas2);
printf("Enter probability for fly ash to be tricalcium aluminate \n");
scanf("%f",&probc3a);
printf("%f\n",probc3a);
printf("Enter probability for fly ash to be calcium chloride \n");
scanf("%f",&probcacl2);
printf("%f\n",probcacl2);
printf("Enter probability for fly ash to be silica \n");
scanf("%f",&probsio2);
printf("%f\n",probsio2);
printf("Enter probability for fly ash to be anhydrite \n");
scanf("%f",&probanh);
printf("%f\n",probanh);

infile=fopen(filein,"r");
outfile=fopen(fileout,"w");

/* Convert to cumulative probabilties */
/* Order must match that used in for loops below */
probcacl2+=probasg;
probsio2+=probcacl2;
probcas2+=probsio2;
probanh+=probcas2;
probc3a+=probanh;
/* Scan each pixel and convert all fly ash pixels to some */
/* component phase */
for(ix=0;ix<SYSSIZE;ix++){
for(iy=0;iy<SYSSIZE;iy++){
for(iz=0;iz<SYSSIZE;iz++){

        fscanf(infile,"%d",&valin);
        valout=valin;
        if(valin==FLYASH){
                valout=INERTID;
                prph=ran1(seed);
                if(prph<probasg){
```

```
                        valout=ASGID;
                    }
                    else if(prph<probcacl2){
                        valout=CACL2;
                    }
                    else if(prph<probsio2){
                        valout=POZZID;
                    }
                    else if(prph<probcas2){
                        valout=CAS2ID;
                    }
                    else if(prph<probanh){
                        valout=ANHYDRITE;
                    }
                    else if(prph<probc3a){
                        valout=C3A;
                    }
                }

                fprintf(outfile,"%d\n",valout);
        }
        }
        }

        fclose(infile);
        fclose(outfile);
}
```

10.4 distfapart.c- Random particle distribution of fly ash

```
#include <stdio.h>
#include <math.h>

/*#########################################################################*/
/* Program distfapart to distribute fly ash phases randomly amongst  */
/* monophase particles                                                */
/* Programmer: Dale P. Bentz                                          */
/* Date: May 1997                                                     */
/*                                                                    */
/*#########################################################################*/

/* Define phase identifiers for fly ash components */
#define FLYASH 25
#define C3A 35
#define CACL2 12
#define ASGID 14
#define INERTID 18
#define POZZID 17
#define ANHYDRITE 20
#define CAS2ID 16
#define SYSSIZE 100
#define NPARTC 12000

int *seed;
#include "ran1.c"

main()
{
        FILE *infile,*partfile,*outfile;
        char filein[80],fileout[80],filepart[80];
        int ix,iy,iz,valin,valout,partin;
        float probasg,probcacl2,probsio2;
        float probc3a,prph,probcas2,probanh;
        static int phase[NPARTC],partid[NPARTC];
        int nseed,count,c1,phnew;
        long int totcnt,ascnt,cacl2cnt,pozzcnt,inertcnt;
        long int anhcnt,cas2cnt,c3acnt;
        long int markc3a,markas,markcacl2,markpozz,markinert,markanh,markcas2;

        ascnt=cacl2cnt=pozzcnt=inertcnt=0;
        cas2cnt=anhcnt=c3acnt=0;
        printf("Enter random number seed value (<0)\n");
```

```
scanf("%d",&nseed);
printf("%d\n",nseed);
seed=(&nseed);

printf("Enter name of file for input \n");
scanf("%s",filein);
printf("%s\n",filein);
printf("Enter name of file for particle IDs \n");
scanf("%s",filepart);
printf("%s\n",filepart);

printf("Enter name of file for output \n");
scanf("%s",fileout);
printf("%s\n",fileout);

printf("Enter total number of fly ash pixels \n");
scanf("%ld",&totcnt);
printf("%ld\n",totcnt);

/* Get user input for phase probabilities (volume fractions */
printf("Enter probability for fly ash to be aluminosilicate glass \n");
scanf("%f",&probasg);
printf("%f\n",probasg);
printf("Enter probability for fly ash to be calcium aluminodisilicate \n");
scanf("%f",&probcas2);
printf("%f\n",probcas2);
printf("Enter probability for fly ash to be tricalcium aluminate \n");
scanf("%f",&probc3a);
printf("%f\n",probc3a);
printf("Enter probability for fly ash to be calcium chloride \n");
scanf("%f",&probcacl2);
printf("%f\n",probcacl2);
printf("Enter probability for fly ash to be silica \n");
scanf("%f",&probsio2);
printf("%f\n",probsio2);
printf("Enter probability for fly ash to be anhydrite \n");
scanf("%f",&probanh);
printf("%f\n",probanh);

/* Determine goal counts for each phase */
markas=(long)(probasg*(float)totcnt);
markpozz=(long)(probsio2*(float)totcnt);
markcacl2=(long)(probcacl2*(float)totcnt);
markanh=(long)(probanh*(float)totcnt);
markcas2=(long)(probcas2*(float)totcnt);
markc3a=(long)(probc3a*(float)totcnt);
```

```
markinert=(long)((1.-probasg-probsio2-probcacl2-probanh-
 probcas2-probc3a)*(float)totcnt);
/* Convert probabilities to cumulative */
/* Order must be the same as in for loop below */
probcacl2+=probasg;
probsio2+=probcacl2;
probanh+=probsio2;
probcas2+=probanh;
probc3a+=probcas2;

infile=fopen(filein,"r");
partfile=fopen(filepart,"r");

for(ix=0;ix<NPARTC;ix++){
        phase[ix]=partid[ix]=0;
}

count=0;
/* First scan-- find each particle and assign phases */
for(ix=0;ix<SYSSIZE;ix++){
for(iy=0;iy<SYSSIZE;iy++){
for(iz=0;iz<SYSSIZE;iz++){

        fscanf(partfile,"%d",&partin);
        fscanf(infile,"%d",&valin);
        if((valin==FLYASH)&&(partid[partin]==0)){
                count+=1;
                partid[partin]=count;

                valout=INERTID;
                do{
                prph=ran1(seed);
                if((prph<probasg)&&(ascnt<markas)){
                        valout=ASGID;
                }
                else if((prph<probcacl2)&&(cacl2cnt<markcacl2)){
                        valout=CACL2;
                }
                else if((prph<probsio2)&&(pozzcnt<markpozz)){
                        valout=POZZID;
                }
                else if((prph<probanh)&&(anhcnt<markanh)){
                        valout=ANHYDRITE;
                }
                else if((prph<probcas2)&&(cas2cnt<markcas2)){
                        valout=CAS2ID;
```

```
                }
                else if((prph<probc3a)&&(c3acnt<markc3a)){
                        valout=C3A;
                }
                }while((valout==INERTID)&&(inertcnt>markinert));
                phase[count]=valout;
        }
        if(valin==FLYASH){
                c1=partid[partin];
                phnew=phase[c1];
                if(phnew==ASGID){
                        ascnt+=1;
                }
                else if(phnew==CACL2){
                        cacl2cnt+=1;
                }
                else if(phnew==POZZID){
                        pozzcnt+=1;
                }
                else if(phnew==ANHYDRITE){
                        anhcnt+=1;
                }
                else if(phnew==CAS2ID){
                        cas2cnt+=1;
                }
                else if(phnew==C3A){
                        c3acnt+=1;
                }
                else if(phnew==INERTID){
                        inertcnt+=1;
                }
       }
}
}
}

fclose(infile);
fclose(partfile);

/* Now distribute phases in second scan */
infile=fopen(filein,"r");
partfile=fopen(filepart,"r");
outfile=fopen(fileout,"w");
for(ix=0;ix<SYSSIZE;ix++){
for(iy=0;iy<SYSSIZE;iy++){
for(iz=0;iz<SYSSIZE;iz++){
```

```
                fscanf(partfile,"%d",&partin);
                fscanf(infile,"%d",&valin);
                valout=valin;
                if(valin==FLYASH){
                        count=partid[partin];
                        valout=phase[count];
                }

                fprintf(outfile,"%d\n",valout);

        }
        }
        }
        fclose(infile);
        fclose(partfile);
        fclose(outfile);
}
```

11 Appendix C- Example input datafile for disreal-new.c

```
-283              random number seed
0                 flag indicating that final microstructure is not to be saved
cimwfa01g.img     filename for file containing input 3-D microstructure
1 3 4 2 5 6       phase IDs assigned to C3S,  C2S, C3A, C4AF, gypsum, and agg.
35                phase ID assigned to C3A in fly ash particles
pcimwfa01.img     filename for file containing particle ID microstructure
0                 number of one pixel particles to add
5000              number of cycles of hydration model to execute
1                 flag indicating hydration is to be under sealed conditions
500               maximum number of diffuion steps per cycle
0.01 9000.        parameters for CH nucleation
0.0002 100000.    parameters for C3AH6 nucleation
0.2 2500.         parameters for FH3 nucleation
200               frequency in cycles to check pore space percolation
5002              frequency in cycles to check total solids percolation
6.0               induction time in hours
25.               initial temperature in degrees Celsius
45.7              activation energy for cement hydration in kJ/mole
83.14             activation energy for pozzolanic reactions in kJ/mole
0.00127           conversion factor to go from cycles to time
0.000             mass fraction of aggregates in mixture proportions
0                 flag indicating hydration is under isothermal conditions
0                 flag indicating primary to pozzolanic CSH conversion is off
```